SemSearch: A Search Engine for the Semantic Web

Yuangui Lei, Victoria Uren, and Enrico Motta

Knowledge Media Institute (KMi), The Open University, Milton Keynes
{y.lei, v.s.uren, e.motta}@open.ac.uk

Abstract. Existing semantic search tools have been primarily designed to enhance the performance of traditional search technologies but with little support for ordinary end users who are not necessarily familiar with domain specific semantic data, ontologies, or SQL-like query languages. This paper presents SemSearch, a search engine, which pays special attention to this issue by providing several means to hide the complexity of semantic search from end users and thus make it easy to use and effective.

1 Introduction

Semantic search promises to produce precise answers to user's queries by taking advantage of the availability of explicit semantics of information in the semantic web. For example, when searching for news stories about *phd students*, with traditional searching technologies, we often could only get news entries in which the term "phd students" appears. Those entries which mention the names of students but do not use the term "phd students" directly will be missed out. Such news entries however are often the ones that the user is interested in. In the context of the semantic web, where the meaning of web content is made explicit, the meaning of the keyword (which is a general concept in the example of phd students) can be figured out. Furthermore, the underlying semantic relations of metadata can be exploited to support the retrieval of related information.

A number of tools have been recently developed [4,3,6,1,5], which enhance the performance of traditional search technologies. While these tools do provide comprehensive support for semantic search, they are however not suitable for ordinary end users who are not necessarily familiar with domain specific semantic data, ontologies, or SQL-like query languages. Some tools [4,1] suffer from the problem of "knowledge overhead", which is requiring end users to be equipped with extensive knowledge on the back-end ontologies, data repositories or the specified sophisticated query language *before* they use them. Some lack support for complex queries, e.g., semantic-based keyword search engines [3,6]. Others [5] heavily rely on the natural language processing techniques that they use.

The semantic search engine we present here, SemSearch, pays special attention to the issue of end user support. It provides several means to address the problems suffered by state-of-art tools. A prototype of the search engine has

S. Staab and V. Svatek (Eds.): EKAW 2006, LNAI 4248, pp. 238–245, 2006.

been implemented and applied in the semantic web portal of our lab[1]. An initial evaluation shows promising results.

The rest of the paper is organized as follows. We begin in Section 2 by presenting an overview of SemSearch. We then explain the Google-like query interface in Section 3. Thereafter, we describe the major steps of the semantic search process in sections 4 and 5. Finally, we conclude with a discussion of our contributions and future work in Section 6.

2 An Overview of SemSearch

One major goal in this work is to hide the complexity of semantic search from end users and to make it easy to use and effective for naive users. To achieve this goal, we identified the following key requirements:

- **Low barrier to access for ordinary end users**. Our semantic search engine should overcome the problem of knowledge overhead and ensure that ordinary end users are able to use it without having to know about the vocabulary or structure of the ontology or having to master a special query language.
- **Dealing with complex queries**. In contrast with existing semantic-based keyword search engines which only answer simple queries, our semantic search engine should allow end users to ask relatively complex queries and provide comprehensive means to handle them.
- **Precise and self-explanatory results**. Our semantic search engine should be able to produce precise results that on the one hand satisfy user queries, and on the other hand are self-explanatory. Thus, ordinary end users can understand the results (e.g. what they are and why they are there) without having to consult the back-end semantic data repositories or their underlying ontologies.
- **Quick response**. Our semantic search engine should provide quick response to user queries, thus encouraging ordinary end users to harvest the benefit of the semantic web technology. This requires that we make the mechanism of semantic search as simple as possible.

To meet these requirements, we chose the keyword-based searching route rather than the natural language question answering route, and deliberately avoided linguistic processing which is a relatively expensive process in terms of search. We overcome the limitation of current keyword-based semantic search engines by supporting a Google-like query interface which supports complex queries in terms of multiple keywords. Figure 1 shows a layered architecture of our semantic search engine. It separates end users from the back-end heterogeneous semantic data repositories by several layers.

- **The Google-like User Interface Layer** allows end users to specify queries in terms of keywords. It extends traditional keyword search languages by

[1] `http://semanticweb.kmi.open.ac.uk:8080/pages/semantic_searhing.jsp/`

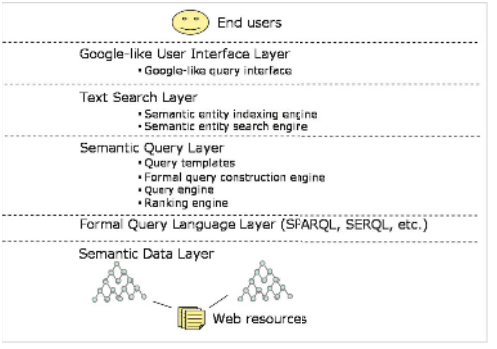

Fig. 1. An overview of the SemSearch architecture

allowing the explicit specification of i) the queried subject and ii) the combination of multiple keywords.

- **The Text Search Layer** interprets user queries by finding out the explicit semantic meanings of the user keywords. Central to this layer are two components: i) a semantic entity index engine, which indexes documents and their associated semantic entities including classes, properties, and individuals; and ii) a semantic entity search engine, which supports the searching of semantic entity matches for the user keywords.
- **The Semantic Query Layer** produces search results for user queries by translating user queries into formal queries. This layer comprises three components, including i) a formal query construction engine, which translates user queries into formal queries, ii) a query engine, which queries the specified meta-data repository using the generated formal queries, and iii) a ranking engine, which ranks the search results according to the degree to which they satisfy the user query.
- **The Formal Query Language Layer** provides a specific formal query language that can be used to retrieve semantic relations from the underlying semantic data layer.
- **The Semantic Data Layer** comprises semantic metadata that are gathered from heterogeneous data sources and are represented in different ontologies.

The search process of SemSearch comprises four major steps:

- **Step1.** Making sense of the user query, which is to find out the semantic meanings of the keywords specified in a user query.
- **Step2.** Translating the user query into formal queries.
- **Step3.** Querying the back-end semantic data repositories using the generated formal queries.
- **Step4.** Ranking the querying results.

Step1 is carried out within the Text Search Layer. The rest of the steps are associated with the Semantic Query Layer.

3 The Google-Like Query Interface

The SemSearch query interface extends traditional keyword search languages by allowing the explicit specification of i) the queried subject which indicates the type of the expected search results, and ii) the combination of keywords. The query interface uses the operator ":" to capture the query subject and the operators "and" and "or" to specify the combination of keywords (apart from the subject keyword). A user query in SemSearch looks like "*subject:keyword1 and/or keyword2 and/or keyword3 ...*".

With this query syntax, the example of "news about phd students" can be easily specified as *news:phd students*, where the term *news* is the query subject and the term *phd students* is a required keyword. More complex queries in which multiple keywords (except the subject keyword) are involved also can be easily specified. For example, when querying for projects in which both Enrico and John participate, the query can be specified as *project:Enrico and John*.

The SemSearch query interface provides a flexible and powerful approach to user query specification. First, it does not require end users to be familiar with any particular ontology, semantic data, or any special query language. Second, it does not confine users to any pre-defined query subjects and values. Further, in contrast with current semantic-based keyword search engines which only accept one keyword as input, this query interface supports the specification of relatively complex queries that specify both multiple keywords and the expected type of results. Finally, the query process is simpler than question answering tools as the search engine does not need to spend time calculating which of the keywords are in a user's query.

4 Interpreting User Queries

As mentioned earlier in Section 2, interpreting user queries is the first step of the search process in SemSearch. The task of this step is to find out the semantic meanings of the keywords specified in user queries so that the search engine knows what the user is looking for and how to satisfy the user query.

From the semantic point of view, one keyword may match i) general concepts (e.g., the keyword "phd students" which matches the concept *phd-student*), ii) semantic relations between concepts, (e.g. the keyword "author" matches the relation *has-author*), or iii) instance entities (e.g., the keyword "Enrico" which matches the instance *Enrico-Motta*, the keyword "chief scientist" which matches the values of the instance *Marc-Eisenstadt* of the property *has-job-title*). The ideal goal of this task is to find out the exact semantic meaning of each keyword. This is however not easy to achieve, as there may be more than one semantic entity which matches a keyword. Thus, we relaxed the goal to that of finding out all the semantic entity matches for each keyword.

For the purpose of finding out semantic entity matches, we used the *labels* of semantic entities as the main search source. The rational for this choice is that, from the user point of view, labels often catch the meaning of semantic

entities in an understandable way. In the case of instances, we also used their short literal values as the search source. So that when the user is searching for "chief scientist", the instance that has such a string as a value of its properties can be reached.

In order to produce fast response, the search engine first indexes all the semantic entities contained in the back-end semantic data repositories, including classes, properties, and instances. It then searches the indexed repository to find out matches for keywords. Thus, two components are developed in the search engine, namely the semantic entity index engine and the semantic entity search engine. As it narrows the search sources to labels and short literals of semantic entities, the search engine is able to find out semantic entity matches for each keyword. These matches are the possible semantic meanings of keywords.

Please note that for the sake of getting quick response, we only use text search to find string matches for user keywords at the moment. We avoid using techniques like WordNet [2] based comparison to find matches. This might cost us some good matches, e.g., losing the match *table* if the user is searching for *desk*. But one to one comparison is time consuming and expensive in real-time scenarios. This is indeed a trade-off as well as a research challenge that we need to address in future.

5 Translating User Queries into Formal Queries

In this step, the search engine takes as input the semantic matches of user search terms and outputs appropriate formal queries. To better understand how to construct formal queries from user queries, we classify user queries into two types: i) simple queries which only comprise two keywords, and ii) complex queries where more than two keywords are involved.

Simple user queries. As the types of semantic entity match combinations are fixed in simple user queries, we developed a set of templates to describe how to retrieve relations between two semantic entities. Among all the combinations, there are three most possible types between two keywords. This is because we can make the assumption that the subject keyword matches a class concept. Figure 2 shows the templates for these combinations[2].

Now let us investigate the first combination where both keywords in a query match classes. The search results are expected to be the instances of the class C_s (i.e. the match of the subject keyword) which have explicitly specified relations with the instances of the class C_k (i.e. the match of the other keyword). For example, when querying for news about "phd students", the expected results are the news entries in which phd students are involved. Further, the search results are also expected to be self-explanatory, e.g., to motivate why certain news entries appear and others do not. Thus, along with the retrieving of news instances, the related phd students and the relations between students and news entries

[2] We used the Sesame SeRQL language (http://www.openrdf.org/) as the formal query language in the SemSearch prototype.

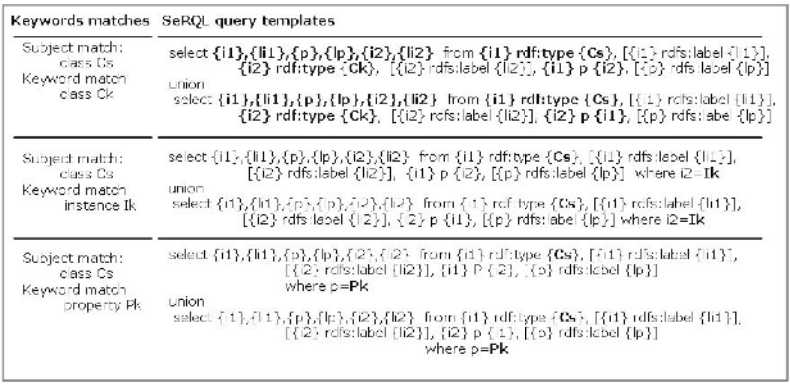

Fig. 2. The SeRQL query templates for two semantic entities

also need to be retrieved. Therefore, the search results of the query *news:phd students* are expected to be triples of *(news, relation, phd-sudent)*.

Please note that there are situations where no class matches could be found for the subject keyword. The focus of user query in such situations varies according to the type of the semantic matches of keywords. We have also developed templates for such queries. Due to the lack of space, please refer to [8] for details.

In the context of simple queries, the task of query formulation is to initiate the template that corresponds to the combinations of the semantic matches of the user keywords. As each keyword may match more than one semantic entity, often more than one query needs to be constructed. More specifically, if the subject keyword matches n_s semantic entities and the other keyword has n_k matches, there are n_s*n_k queries that need to be constructed. This problem becomes more acute when there are many keywords involved in the user query. We will discuss how to reduce the number of formal queries in the following.

Complex user queries. For complex queries (which involve more than two keywords), the search engine needs to combine the semantic matches of each keyword together and construct queries for each of the combinations. A key operational problem is that in real world situations there can be a large number of matches and hence much more combinations.

For keywords k_1, k_2, ..., k_n, suppose that the number of the semantic matches of the keyword k_i is n_i. There will be n_1*n_2*...*n_n (which can be represented as $\prod_{i=1}^{n} n_i$) different combinations when considering all the keywords as required ones. Each combination of the matches corresponds to a RDF-based formal query. Apart from considering all the keywords as required ones, the search engine also needs to investigate the combinations where one or more keywords are left out, in order to produce complete result sets to end users. The total number can become huge when i) there are many keywords involved and ii) some keywords are very generic and thus have many matches.

Rules are therefore needed to reduce the number of matches for each keyword. We used several heuristic rules, including i) the subject keyword always

matches class entities when there are more than two keywords involved in the user query, ii) choosing the closest entity matches to the keyword as possible, and iii) choosing the most specific class match among the class matches. These rules can significantly reduce the number of entity matches.

For each combination, a formal query is constructed. In SeRQL, a formal query often comprises three building blocks: the *head* block, which describes what needs to be retrieved, the *body* block, which describes how, and the *condition* block, which expresses conditions. In addition, in order to cover relations of two entities in both directions, the query also comprises a *union* block. Figure 3 shows the construction algorithm. As shown in the figure, the construction of all these blocks depends on the type (i.e. class, property, or instance) of the semantic entity match of each keyword contained in a combination of keywords' matches. Please refer to [8] for details.

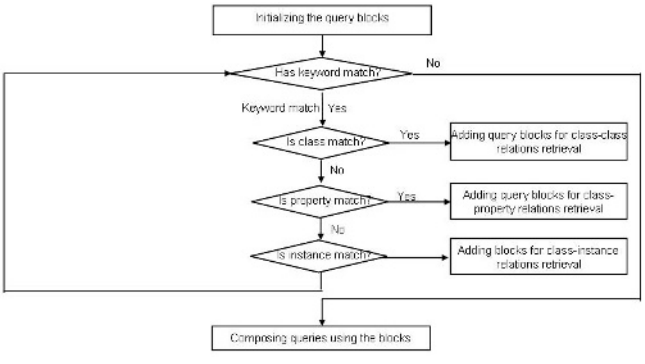

Fig. 3. The algorithm of formal query construction for complex user queries

6 Conclusions and Future Work

The core observation that underlies this paper is that, in the case of semantic search that promises to produce precise answers to user queries, it is important to ensure that it is easy to use and effective for ordinary end users who are not necessarily familiar with domain specific semantic data, ontologies, or SQL-like query languages. Our semantic search engine, SemSearch, provides several means to address this issue. A prototype has been implemented based on the Sesame RDF query engine and Lucene text search engine [3]. The prototype has been applied to the semantic web portal of our lab (KMi) and the 3rd European Semantic Web Conference (ESWC06) [4]. Figure 4 shows a screenshot of the search results of the query example *news:phd students* in the KMi application.

Future work will focus on i) developing comprehensive means to perform semantic matching between keywords and semantic entities and ii) extending the

[3] `http://lucene.apache.org/`
[4] `http://search.eswc06.org/`

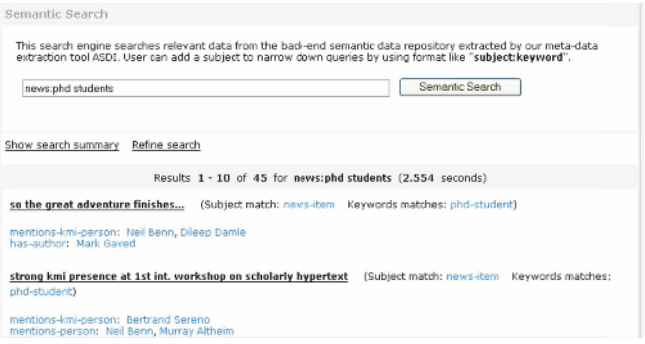

Fig. 4. A screenshot of the search results of the query example *news:phd students*

search engine to a tool that could guide end users to build up complex queries step by step by using the component-based approach presented in [7].

Acknowledgements

We wish to thank Marta Sabou for her valuable comments on this paper. This work was funded by the Advanced Knowledge Technologies Interdisciplinary Research Collaboration (IRC) GR/N15764/01 and the X-Media project (www.x-media-project.org) under EC grant number IST-FP6-026978.

References

1. O. Corby, R. Dieng-Kuntz, and C. Faron-Zucker. Querying the Semantic web with Corese Search Engine. In *Proceedings of 15th ECAI/PAIS, Valencia (ES)*, 2004.
2. C. Fellbaum. *WORDNET: An Electronic Lexical Database*. MIT Press, 1998.
3. R. Guha, R. McCool, and E. Miller. Semantic Search. In *Proceedings of the 12th international conference on World Wide Web*, pages 700–709, 2003.
4. J. Heflin and J. Hendler. Searching the Web with SHOE. In *Proceedings of the AAAI Workshop on AI for Web Search*, pages 35 – 40. AAAI Press, 2000.
5. V. Lopez, M. Pasin, and E. Motta. AquaLog: An Ontology-portable Question Answering System for the Semantic Web. In *Proceedings of European Semantic Web Conference (ESWC 2005)*, 2005.
6. C. Rocha, D. Schwabe, and M. de Aragao. A Hybrid Approach for Searching in the Semantic Web. In *Proceedings of the 13th International World Wide Web Conference*, 2004.
7. V. Uren and E. Motta. Semantic search components: a blueprint for effective query language interfaces. In *The International Conference on Knowledge Engineering and Knowledge Management (EKAW 2006)*, October 2006.
8. Y.Lei, V. Uren, and E.Motta. SemSearch: A Search Engine for the Semantic Web. Technical Report kmi-06-11, Knowledge Media Institute, the Open University, http://kmi.open.ac.uk/publications/pdf/semsearch_paper.pdf, 2006.